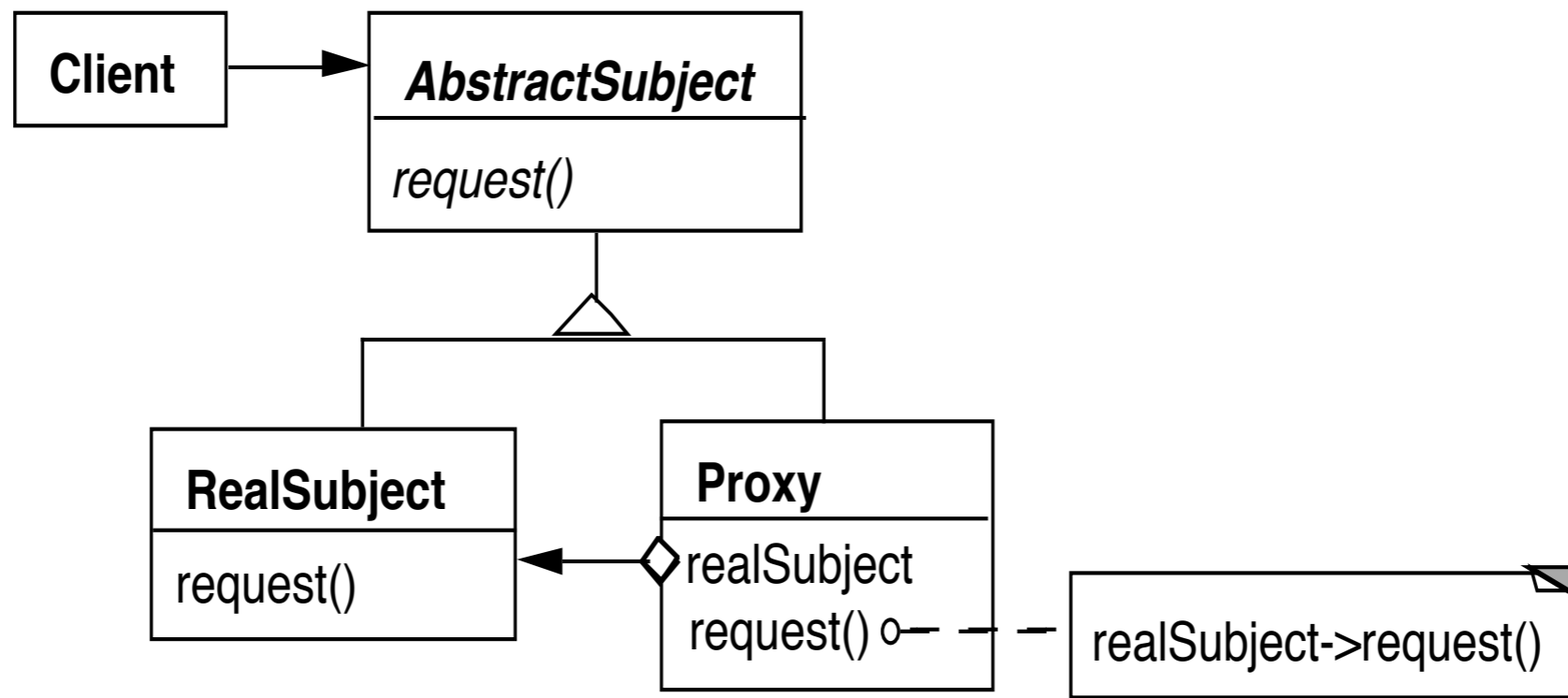


CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2014
Doc 12 Proxy, Coupling
Mar 11, 2014

Copyright ©, All rights reserved. 2014 SDSU & Roger Whitney, 5500
Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent ([http://
www.opencontent.org/opl.shtml](http://www.opencontent.org/opl.shtml)) license defines the copyright on this
document.

Proxy (Surrogate)

a person authorized to act on behalf of another



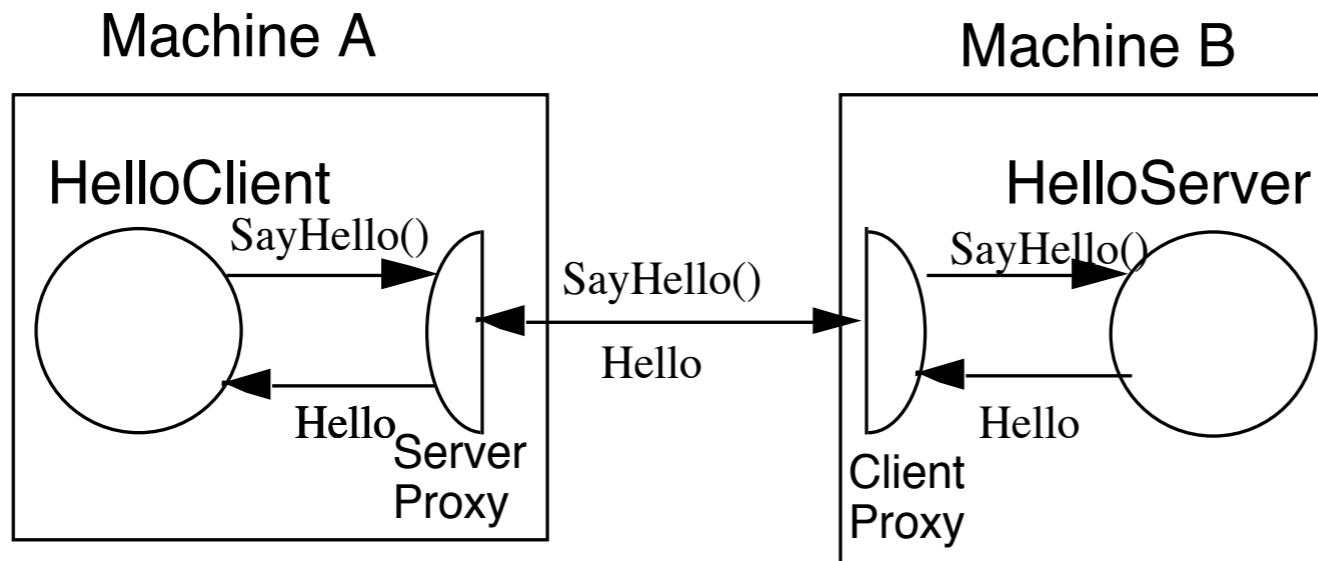
```

class Proxy {
    AbstractSubject realSubject;

    public Foo service(Bar x ) {
        return realSubject(x);
    }
}
  
```

Why do it?

Remote Proxy



```
String server = getHelloHostAddress( args);  
Hello proxy = (Hello) Naming.lookup( server );  
String message = proxy.sayHello();  
System.out.println( message );
```

More General Proxy

```
class Proxy {  
    AbstractSubject realSubject;  
  
    public Foo service(Bar x ) {  
        some preprocessing  
        result = realSubject(x);  
        some postprocessing  
    }  
}
```

Virtual Proxy

Creates/accesses expensive objects on demand

O-R Mapping Layers

Java's Synchronized List

```
ArrayList notSafe = new ArrayList();  
List threadSafe = Collections.synchronizedList(notSafe);
```

```
static class SynchronizedList {  
    List list;  
    public Object get(int index) {  
        synchronized(mutex) {return list.get(index);}  
    }  
}
```


Java's Unmodifiable List

```
ArrayList notSafe = new ArrayList();  
List noChange = Collections.unmodifiableList(notSafe);
```

```
static class UnmodifiableList {  
    List list;  
    public Object get(int index) { return list.get(index);}   
  
    public Object set(int index, Object element) {  
        throw new UnsupportedOperationException();  
    }  
}
```

Proxy or Decorator?

```
ArrayList notSafe = new ArrayList();  
List noChange = Collections.unmodifiableList(notSafe);  
List threadSafe = Collections.synchronizedList(noChange);
```



Proxy verses Decorator

"Decorators can have similar implementations as proxies"

Proxy controls access to an object

Decorator adds one or more responsibilities to an object

Smalltalk Proxy Trick

Object subclass: #Proxy

instanceVariableNames: 'target '

classVariableNames: "

poolDictionaries: "

category: 'Whitney-Examples'

Class Method

on: anObject

^super new target: anObject

Instance Methods

doesNotUnderstand: aMessage

^target

perform: aMessage selector

withArguments: aMessage arguments

target: anObject

target := anObject

| wrapper |

wrapper := Proxy on: Transcript.

wrapper open.

wrapper show: 'Hi mom'.

| wrapper |

wrapper := Proxy on: 3.

wrapper + 5.

| wrapper |

wrapper := Proxy on: 'Hi '.

wrapper , ' mom'.

Java Proxy Trick

`java.lang.reflect.Proxy`

Can build a proxy object to interfaces

Interface and Implementation

```
public interface BarInterface {  
    public String hello();  
  
    public int foo();  
}
```

```
public class Bar implements BarInterface {  
  
    public String hello() {  
        return "Hello world";  
    }  
  
    public int foo() {  
        return 1;  
    }  
}
```

Invocation Handler

```
public class BarInvocationHandler implements InvocationHandler {  
  
    private Bar realSubject;  
    private int methodCallCount = 0;  
  
    public BarInvocationHandler(Bar target) {  
        this.realSubject = target;  
    }  
}
```

Invocation Handler

```
public Object invoke(Object proxy, Method method, Object[] args)
    throws Throwable {
    methodCallCount++;
    if (Object.class == method.getDeclaringClass()) {
        String name = method.getName();
        if ("equals".equals(name)) {
            return proxy == args[0];
        } else if ("hashCode".equals(name)) {
            return System.identityHashCode(proxy);
        } else if ("toString".equals(name)) {
            return realSubject.toString();
        } else {
            throw new IllegalStateException(String.valueOf(method));
        }
    }
    if (method.getName() == "hello")
        return method.invoke(realSubject, args) + " from proxy";
    return method.invoke(realSubject, args);
}
```


Using Proxy

```
public static void main(String[] args) {
    Bar realSubject = new Bar();
    BarInvocationHandler handler = new BarInvocationHandler(realSubject);
    BarInterface proxy = (BarInterface) Proxy.newProxyInstance(
        Bar.class.getClassLoader(), new Class[] { BarInterface.class },
        handler);
    System.out.println(proxy.hello());           //Hello world from proxy
    System.out.println("Foo method: " + proxy.foo()); //Foo method: 1

    if (proxy instanceof BarInterface)
        System.out.println("is instance of BarInterface"); //is instance of BarInterface
    if (proxy instanceof Bar)
        System.out.println("is instance of Bar");
    System.out.println("Proxy class: " + proxy.getClass()); //Proxy class: class $Proxy0
    System.out.println(proxy.toString()); // Bar@10f0f6ac
    System.out.println("Count " + handler.methodCallCount()); //Count 3
}
```

Coupling

In the Beginning

Parnas (72) KWIC (Simple key word in context) experiment

Read lines of words

Output all circular shifts of all lines in alphabetical order

Circular shift

remove first word of line and add it to the end of the line

KWIC Solutions

Solution 1

Each major step in processing is a module

Create flowchart and make each major part a module

Solution 2

Modules based on design decisions

List design decisions that are
Difficult
Likely to change

Each module should hide a design decision

Solution 1

More complex
Harder to understand
Much harder to modify

Metrics for Quality

Coupling

Strength of interaction between objects in system

Cohesion

Degree to which the tasks performed by a single module are functionally related

Coupling

Measure of the interdependence among modules

"Unnecessary object coupling needlessly decreases the reusability of the coupled objects"

"Unnecessary object coupling also increases the chances of system corruption when changes are made to one or more of the coupled objects"

Design Goal

The interaction or other interrelationship between any two components at the same level of abstraction within the system be as weak as possible

Types of Modular Coupling

In order of desirability

Data Coupling (weakest – most desirable)

Control Coupling

Global Data Coupling

Internal Data Coupling (strongest – least desirable)

Content Coupling (Unrated)

Data Coupling

Output from one module is the input to another
Using parameter lists to pass items between routines

Common Object Occurrence

Object A passes object X to object B
Object X and B are coupled
A change to X's interface may require a change to B

Example

```
class ObjectBClass{
    public void message( ObjectXClass X ){
        // code goes here
        X.doSomethingForMe( Object data );
        // more code
    }
}
```


Data Coupling

Problem

Object A passes object X to object B

X is a compound object

Object B must extract component object Y out of X

B, X, internal representation of X, and Y are coupled

```
public class HiddenCoupling {  
    public bar someMethod(SomeType x) {  
        AnotherType y = x.getY();  
        y.foo();  
        blah;  
    }  
}
```

Example – Sorting

How to write a general purpose sort

Sort the same list by

ID

Name

Grade

```
class StudentRecord {  
    Name lastName;  
    Name firstName;  
    long ID;  
  
    public Name getLastName() { return  
lastName; }  
    // etc.  
}
```

```
SortedList cs635 = new SortedList();  
StudentRecord newStudent;  
//etc.  
cs535.add ( newStudent );
```

Attempt 1

```
class SortedList
{
    Object[] sortedElements = new Object[ properSize ];

    public void add( StudentRecord X )
    {
        // coded not shown
        Name a = X.getLastName();
        Name b = sortedElements[ K ].getLastName();
        if ( a.lessThan( b ) )
            // do something
        else
            // do something else
        }
    }
}
```

Attempt 2

```
class SortedList{
    Object[] sortedElements = new Object[ properSize ];

    public void add( StudentRecord X ) {
        // coded not shown
        if ( X.lessThan( sortedElements[ K ] ) )
            // do something
        else
            // do something else
    }
}

class StudentRecord{
    private Name lastName;
    private long ID;

    public boolean lessThan( Object compareMe ) {
        return lastName.lessThan( compareMe.lastName );
    }
    etc.
}
```

Attempt 3

```
interface Comparable {
    public boolean lessThan( Object compareMe );
    public boolean greaterThan( Object compareMe );
    public boolean equal( Object compareMe );
}

class StudentRecord implements Comparable {
    blah
    public boolean lessThan( Object compareMe ) {
        return lastName.lessThan( ((Name)compareMe).lastName );
    }
}

class SortedList {
    Object[] sortedElements = new Object[ properSize ];

    public void add( Comparable X ) {
        // coded not shown
        if ( X.lessThan( sortedElements[ K ] )
            // do something
        else
            // do something else
        }
    }
}
```

Attempt 4

```
interface Comparing {
    public boolean lessThan( Object a, Object b );
    public boolean greaterThan( Object a, Object b );
    public boolean equal( Object a, Object b );
}

class StudentNameComparing implements Comparing {
    public boolean lessThan( Object a, Object b ) {
        return ((Student) a).lastName() < ((Student) b).lastName(); }
    etc.
}

class SortedList {
    Object[] sortedElements = new Object[ properSize ];
    Comparing comparer;
    public SortedList(Comparing y) {comparer = y;}

    public void add( Object X ) {
        // coded not shown
        if ( comparer.lessThan( sortedElements[ K ], X )
            // do something
        else
            // do something else
        }
    }
}
```

C++ Version

```
typedef int (*compareFun ) ( StudentRecord, StudentRecord );
class SortedList {
    StudentRecord[] sortedElements =
        new StudentRecord[ properSize ];

    int (*compare ) ( StudentRecord, StudentRecord );

    public setCompare( compairFun newCompare )
        { compare = newCompare; }

    public void add( StudentRecord X ) {
        // coded not shown
        if ( compare( X, sortedElements[ K ] ) )
            // code not shown
    }
}

int compareID( StudentRecord a, StudentRecord b ) { // code not shown }

int compareName( StudentRecord a, StudentRecord b ) { // code not shown }

SortedList myList = new SortedList();
myList.setCompair( compareID );
```

Functor Pattern

Functors are functions that behave like objects

They serve the role of a function, but can be created, passed as parameters, and manipulated like objects

A functor is a class with a single member function

Types of Coupling

Data Coupling (weakest – most desirable)

Control Coupling

Global Data Coupling

Internal Data Coupling (strongest – least desirable)

Content Coupling (Unrated)

Control Coupling

Passing control flags between modules so that one module controls the sequencing of the processing steps in another module

Common Object Occurrence

A sends a message to B

B uses a parameter of the message to decide what to do

```
class Lamp {  
    public static final ON = 0;  
  
    public void setLamp( int setting ) {  
        if ( setting == ON )  
            //turn light on  
        else if ( setting == 1 )  
            // turn light off  
        else if ( setting == 2 )  
            // blink  
    }  
}
```

```
Lamp reading = new Lamp();  
reading.setLamp( Lamp.ON );  
reading.setLamp)( 2 );
```

Cure

Decompose the operation into multiple primitive operations

```
class Lamp {  
    public void on() { //turn light on }  
    public void off() { //turn light off }  
    public void blink() { //blink }  
}
```

```
Lamp reading = new Lamp();  
reading.on();  
reading.blink();
```

Is this Control Coupling

```
class BankAccount {  
    public void withdrawal(Float amount) {  
        balance = balance - amount;  
    }  
etc.
```

Is this Control Coupling

```
class BankAccount {  
    public void withdrawal(Float amount) {  
        if (balance < amount)  
            this.bounceThisCheck();  
        else  
            balance = balance - amount;  
    }  
etc.
```

What if the Lamp had 50 settings?

Control Coupling

Common Object Occurrence

A sends a message to B

B returns control information to A

Example: Returning error codes

```
class Test {  
    public int printFile( File toPrint ) {  
        if ( toPrint is corrupted )  
            return CORRUPTFLAG;  
        blah blah blah  
    }  
}
```

```
Test when = new Test();  
int result = when.printFile( popQuiz );  
if ( result == CORRUPTFLAG )  
    blah  
else if ( result == -243 )
```

Cure – Use Exceptions

How does this reduce coupling?

```
class Test {  
    public int printFile( File toPrint ) throws PrintException  
    {  
        if ( toPrint is corrupted )  
            throws new PrintException();  
        blah blah blah  
    }  
}
```

```
try {  
    Test when = new Test();  
    when.printFile( popQuiz );  
}  
catch ( PrintException printError ) {  
    do something  
}
```

Types of Coupling

Data Coupling (weakest – most desirable)

Control Coupling

Global Data Coupling

Internal Data Coupling (strongest – least desirable)

Content Coupling (Unrated)

Global Data Coupling

Global Data is evil

Global Data Coupling

What are the following?

System.out

Integer.MAX_VALUE

Types of Global Data Coupling in increasing order of "badness"

Make a reference to a specific external object

Make a reference to a specific external object, and to methods in the external object

A component of an object-oriented system has a public interface which consists of items whose values remain constant throughout execution, and whose underlying structures/ implementations are hidden

A component of an object-oriented system has a public interface which consists of items whose values remain constant throughout execution, and whose underlying structures/ implementations are not hidden

A component of an object-oriented system has a public interface which consists of items whose values do not remain constant throughout execution, and whose underlying structures/implementations are hidden

A component of an object-oriented system has a public interface which consists of items whose values do not remain constant throughout execution, and whose underlying structures/implementations are not hidden

Types of Coupling

Data Coupling (weakest – most desirable)

Control Coupling

Global Data Coupling

Internal Data Coupling (strongest – least desirable)

Content Coupling (Unrated)

Internal Data Coupling

One module directly modifies local data of another module

Common Object Occurrences

C++ Friends

Smalltalk reflection

Java reflection

Internal Data Coupling

Implement a debugger without using internal data coupling

Types of Coupling

Data Coupling (weakest – most desirable)

Control Coupling

Global Data Coupling

Internal Data Coupling (strongest – least desirable)

Content Coupling (Unrated)

Lexical Content Coupling

Some or all of the contents of one module are included in the contents of another

Common Object Occurrence

C/C++ header files

Decrease coupling by

Restrict what goes in header file

C++ header files should contain only class interface specifications